图文 98 NameServer最终是如何启动Netty网络通信服务器的?

详情 评论

NameServer最终是如何启动Netty网络通信服务器的?

继《从零开始带你成为JVM实战高手》后，阿里资深技术专家携新作再度出山，重磅推荐：

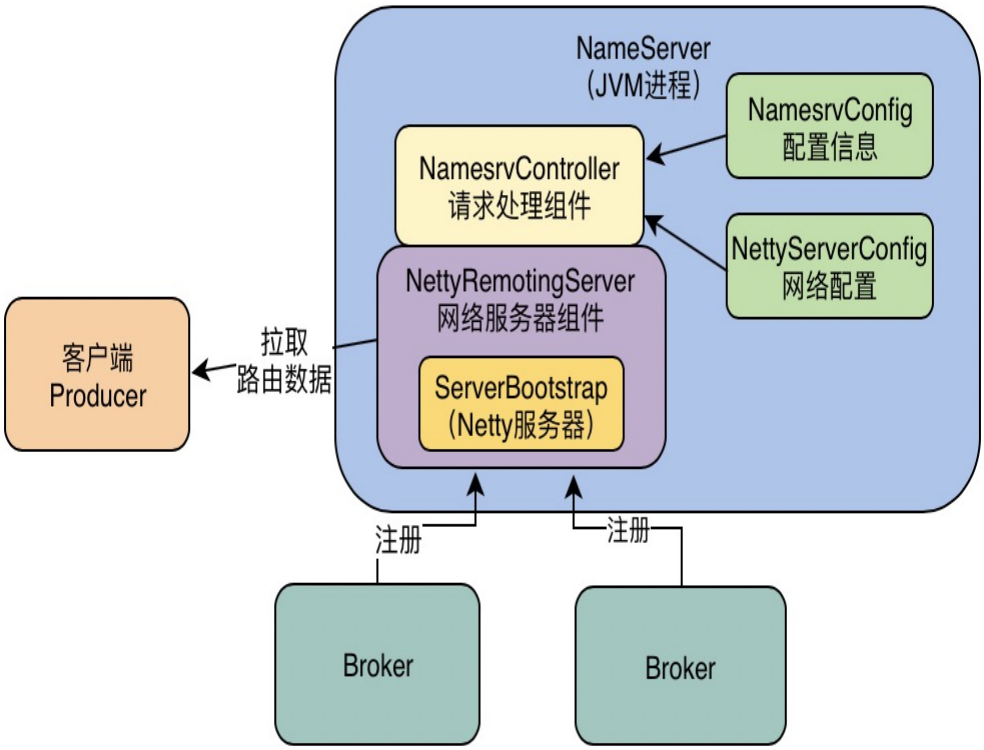
（点击下方蓝字试听）

《从零开始带你成为MySQL实战优化高手》

1、NamesrvController初始化过程的一些遗留代码

上次我们其实整体源码是分析到NamesrvController的initialize()方法，他在进行初始化，然后讲到他初始化了NettyRemotingServer，其中包含了一个Netty API开发的ServerBootstrap，说白了，就是一个网络服务器

我们看下下面的图，已经讲解的很明白了。



接着我们简单看下NamesrvController.initialize()方法遗留下来的还没讲的一些源码，那些源码暂时其实对我们来说都不是太重要，我稍微给大家讲一下就行了。

```
public boolean initialize() {

    // 这个就不用多说了，一看就是在加载kv配置之类的东西，暂时不用关注
    this.kvConfigManager.load();

    // 这个是上一次分析的最重要的一行代码，初始化了Netty服务器
    this.remotingServer = new NettyRemotingServer(
        this.nettyServerConfig,
        this.brokerHousekeepingService);

    // 这个其实还是有一点重要的，他其实就是Netty服务器的工作线程池
    this.remotingExecutor = Executors.newFixedThreadPool(
        nettyServerConfig.getServerWorkerThreads(),
        new ThreadFactoryImpl("RemotingExecutorThread_"));

    // 下面这个，其实就是把工作线程池给了Netty服务器
    this.registerProcessor();

    // 下面这个就是启动了一个后台线程，执行定时任务的
    // 其实你看下面的那行代码，scanNotActiveBroker()，就知道他是干嘛的
    // 他就是定时扫描哪些Broker没发送心跳，就知道他挂了
    // 暂时知道有这么个后台线程就行了，后续等我们需要的时候回来看
    this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {

        @Override
        public void run() {
            NamesrvController.this.routeInfoManager.scanNotActiveBroker();
        }
    }, 5, 10, TimeUnit.SECONDS);

    // 这个也是启动一个后台线程执行定时任务
    // 不过他就很简单了，他就是定时打印kv配置信息，可以忽略他了
    this.scheduledExecutorService.scheduleAtFixedRate(new Runnable() {
```

```
34
35          @Override
36          public void run() {
37              NamesrvController.this.kvConfigManager.printAllPeriodically();
38          }
39      }, 1, 10, TimeUnit.MINUTES);
40
41      // 下面这个其实是FileWatch相关的，很多人肯定看不懂他干嘛的
42      // 其实看源码的时候，碰到这种代码是很正常的，看不懂就看不懂了，不用深究他
43      // 你就直接忽略和跳过下面的代码就行了
44      if (TlsSystemConfig.tlsMode != TlsMode.DISABLED) {
45
46          // Register a listener to reload SslContext
47          try {
48              fileWatchService = new FileWatchService(
49
50                  new String[] {
51                      TlsSystemConfig.tlsServerCertPath,
52                      TlsSystemConfig.tlsServerKeyPath,
53                      TlsSystemConfig.tlsServerTrustCertPath
54                  },
55
56                  new FileWatchService.Listener() {
57
58                      boolean certChanged, keyChanged = false;
59                      @Override
60                      public void onChanged(String path) {
61
62                          if (path.equals(TlsSystemConfig.tlsServerTrustCertPath)) {
63                              log.info("...");
64                              reloadServerSslContext();
65                          }
66
67                          if (path.equals(TlsSystemConfig.tlsServerCertPath)) {
68                              certChanged = true;
69                          }
70
71                          if (path.equals(TlsSystemConfig.tlsServerKeyPath)) {
72                              keyChanged = true;
73                          }
74
75                          if (certChanged && keyChanged) {
76                              log.info("...");
77                              certChanged = keyChanged = false;
78                              reloadServerSslContext();
79                          }
80                      }
81
82                      private void reloadServerSslContext() {
83                          ((NettyRemotingServer) remotingServer).loadSslContext();
84                      }
85                  });
86          } catch (Exception e) {
87              log.warn("...");
88          }
89      }
90      return true;
91  }
```

通过上面的源码分析，你会发现NamesrvController.initialize()方法，最核心的还是初始化Netty网络服务器，其他的就是启动了后台线程执行定时任务还重要一些，但是暂时我们还不用关注他，其他的代码你给忽略了也是可以的。

希望大家在跟着我逐步的分析RocketMQ核心源码的时候，也能够慢慢的掌握我分析源码的思路，你要明白，哪些是需要你重点关注的，哪些你可以暂时放着后续再来看，哪些是你可以干脆给忽略掉了。

2、回到start(controller)方法里看看

接着我们回到start(controller)方法里看看，大家看一下。

```
public static NamesrvController start(
    final NamesrvController controller) throws Exception {

    if (null == controller) {
        throw new IllegalArgumentException(
            "NamesrvController is null");
    }
    boolean initResult = controller.initialize();
    if (!initResult) {
        controller.shutdown();
        System.exit(-3);
    }
    Runtime.getRuntime().addShutdownHook(
        new ShutdownHookThread(log, new Callable<Void>() {

        @Override
        public Void call() throws Exception {
            controller.shutdown();
            return null;
        }
    }));
    controller.start();
    return controller;
}
```

上面的controller.initialize()初始化这块代码我们实际上已经看完了，知道他已经初始化了Netty服务器出来，然后接着我们往下看他通过Runtime类注册了一个JVM关闭时候的shutdown钩子，就是JVM关闭的时候会执行上述注册的回调函数。

那个回调函数里执行了NamesrvController.shutdown()方法，其实我们都不用看里面的代码，都会知道，这里无非都是一些关闭Netty服务器的释放网络资源和线程资源的一些代码，如果大家一定要看，那我们看一下下面的代码。

```
public void shutdown() {

    this.remotingServer.shutdown();
    this.remotingExecutor.shutdown();
    this.scheduledExecutorService.shutdown();

    if (this.fileWatchService != null) {
        this.fileWatchService.shutdown();
    }
}
```

感觉如何？是不是发现他就是在关闭NettyRemotingServer释放网络资源，然后关闭RemotingExecutor就是释放Netty服务器的工作线程池的资源，还有关闭ScheduledExecutorService就是释放执行定时任务的后台线程资源。

其实这里最关键的一行代码是：controller.start()。说白了，他已经初始化了Netty服务器了，但是现在还没启动，没启动的话，Netty服务器就不会监听9876这个默认的端口号，那么NameServer就什么也干不了。

所以此时，他必须要对NamesrvController组件做一个启动操作，这样的话，就可以把他内部的Netty服务器给启动了。

3、Netty服务器是如何启动的?

接着我们进入controller.start()方法内部看看，如下。

```
public void start() throws Exception {

    this.remotingServer.start();

    if (this.fileWatchService != null) {
        this.fileWatchService.start();
    }
}
```

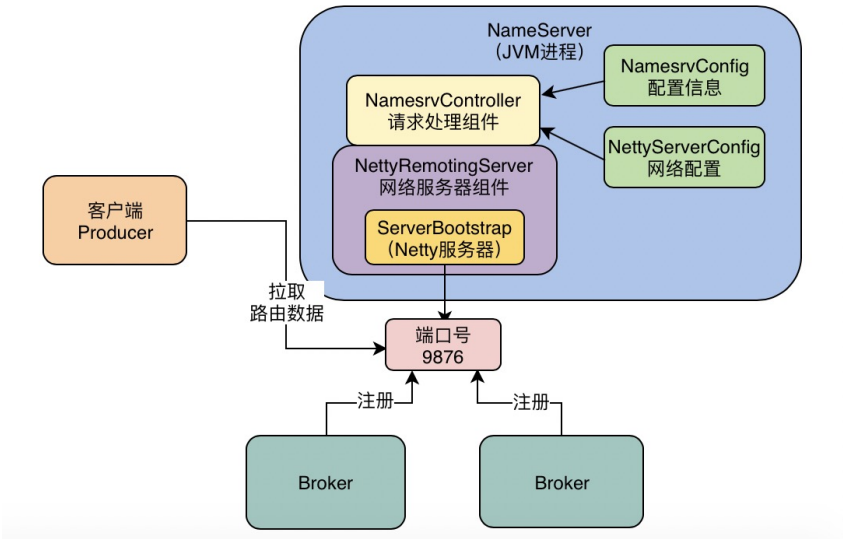
其实这里就很清晰了，这个NamesrvContorller启动，核心就是在启动NettyRemotingServer，也就是Netty服务器。在remotingServer.start()方法里，有很多的代码，我给大家逐步的分析各个片段。

```
// 很多朋友可能对Netty开发不太熟悉，但是其实下面这里
// 主要核心就是基于Netty的API去配置和启动一个Netty网络服务器
ServerBootstrap childHandler =

    // 下面这个地方，基于ServerBootstrap的group方法
    // 对Netty服务器进行各种网络上的配置
    // 这些配置就不去细讲了，因为都是跟Netty相关的
    this.serverBootstrap.group(
     this.eventLoopGroupBoss,
     this.eventLoopGroupSelector).channel(useEpoll() ?
                                          EpollServerSocketChannel.class :
                                          NioServerSocketChannel.class)

     .option(ChannelOption.SO_BACKLOG, 1024)
     .option(ChannelOption.SO_REUSEADDR, true)
     .option(ChannelOption.SO_KEEPALIVE, false)
     .childOption(ChannelOption.TCP_NODELAY, true)
     .childOption(ChannelOption.SO_SNDBUF,
                  nettyServerConfig.getServerSocketSndBufSize())
     .childOption(ChannelOption.SO_RCVBUF,
                  nettyServerConfig.getServerSocketRcvBufSize())

     .localAddress(new InetSocketAddress(this.nettyServerConfig.getListenPort()))
```

```
        // 下面这个地方，说白了，就是设置了一大堆的网络请求处理器
        // 只要Netty服务器收到一个请求，那么就会依次使用下面的处理器来处理请求
        // 比如说handShakeHandler可能就是负责连接握手

        // NettyDecoder是负责编码解码的，IdleStateHandler是负责连接空闲管理的
        // connectionManageHandler是负责网络连接管理的
        // serverHandler是负责最关键的网络请求的处理的
        .childHandler(new ChannelInitializer<SocketChannel>() {
            @Override
            public void initChannel(SocketChannel ch) throws Exception {
                ch.pipeline()
                    .addLast(defaultEventExecutorGroup,
                            HANDSHAKE_HANDLER_NAME,
                            handshakeHandler)
                    .addLast(defaultEventExecutorGroup,
                        encoder,
                        new NettyDecoder(),
                        new IdleStateHandler(0, 0, nettyServerConfig.
                                                getServerChannelMaxIdleTimeSeconds()),
                        connectionManageHandler,
                        serverHandler
                    );
            }
        });
    try {
       // 下面的代码，其实就是启动Netty服务器了
       // 那个bind方法，说白了，其实就是绑定和监听一个端口号
       ChannelFuture sync = this.serverBootstrap.bind().sync();
       InetSocketAddress addr = (InetSocketAddress) sync.channel().localAddress();
       this.port = addr.getPort();
    } catch (InterruptedException e1) {
       ...
    }
```

接着回看上面的一行代码：.localAddress(new InetSocketAddress(this.nettyServerConfig.getListenPort()))。这行代码，其实就是设置了Netty服务器要监听的端口号，默认就是9876

因此到此为止，你可以理解为Netty服务器启动了，开始监听端口号9876了，此时我们看下面的图，图里就展示出了Netty服务器监听端口号的这个示意。



4、总结

到此为止，我们已经初步了解了NameServer是如何启动的了，了解到他最核心的就是基于Netty实现了一个网络服务器，然后监听默认的9876端口号，可以接收Broker和客户端的网络请求。

接着明天开始我们就要研究一下NameServer启动好之后，Broker是如何启动的，如何向NameServer进行注册，如何进行心跳，NameServer是如何管理Broker的。

5、今天作业

今天给大家留的小作业，就是仔细看看NettyRemotingServer的start()方法，仔细看一下里面是如何基于Netty API实现一个网络服务器的配置和启动的。请大家认真完成作业，有什么问题或者心得，欢迎在评论区留言

End

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情留言提问，我会逐一答疑

如何加群：购买狸猫技术窝专栏的小伙伴都可以加入狸猫技术交流群，一个非常纯粹的技术交流的地方

具体加群方式，请参见目录菜单下的文档：《付费用户如何加群》（**购买后可见**）